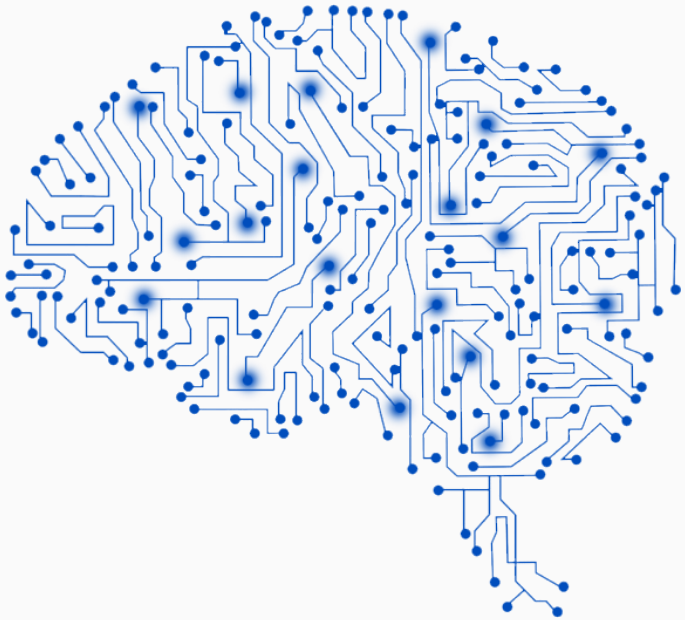
Master 1 Statistique & Data Science, Ingénierie Mathématique

Apprentissage pour l'image
Machine learning for image processing

Final slides

Emile Pierret

Jeudi 04 avril 2024



Plan of the course: How to learn from examples

3 main ingredients

1. Training set / examples (Données d'entraînement):

$$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$$

2. Machine or model (Modèle):

$$\boldsymbol{x} \rightarrow \underbrace{f(\boldsymbol{x}; \theta)}_{\text{function / algorithm}} \rightarrow \underbrace{\boldsymbol{y}}_{\text{prediction}}$$

θ: parameters of the model

3. Loss, cost, objective function / energy (Fonction de coût):

$$\underset{\theta}{\operatorname{argmin}}\ E(\theta; \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$$

Vocabulaire/anglicissismes

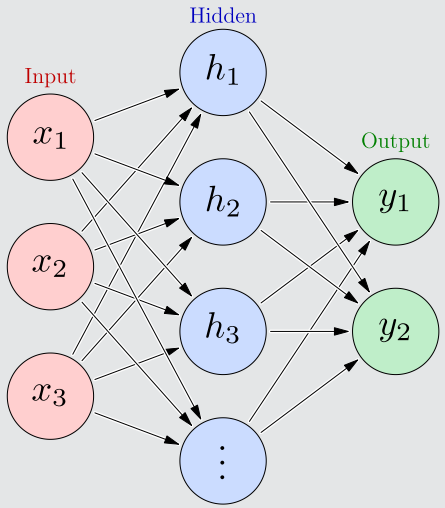
- dataset = ensemble de données
- label = étiquette
- training dataset = ensemble de données d'entraînement
- test dataset = ensemble de données de test
- loss = coût (à minimiser)
- features = caractéristiques
- network = réseau
- Convolutional Neural Network (CNN) = réseau de neurones convolutif
- layer = couche
- hidden layer = couche cachée
- input = entrée
- output = sortie

Un réseau de neurones est une fonction

Universal Approximation Theorem

(Hornik et al, 1989; Cybenko, 1989)

Toute fonction $f : K \subset \mathbb{R}^N \to \mathbb{R}^K$ peut être approximée uniformément par un réseau avec une seule couche cachée avec un nombre suffisant de neurones.



$$\boldsymbol{h} = g(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$$
$$\boldsymbol{y} = \boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2$$

- Le théorème ne renseigne pas sur l'ordre de grandeur du nombre de neurones nécessaire.
- Il n'y a pas de garantie qu'on puisse concrètement optimiser un tel réseau pour toute fonction.

Le cadre du cours:

Nous souhaitons construire des réseaux de neurones pour la classification

Le cadre du cours:

Nous souhaitons construire des réseaux de neurones pour la classification dans un cadre **d'apprentissage supervisé**

Le cadre du cours:

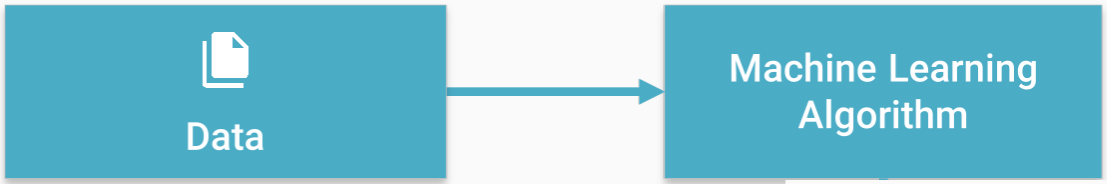
Nous souhaitons construire des réseaux de neurones pour la classification dans un cadre **d'apprentissage supervisé**

Définition (apprentissage supervisé)

L'apprentissage supervisé consiste à superviser l'apprentissage de la machine en lui montrant des exemples (des données) de la tâche qu'elle doit réalisée.

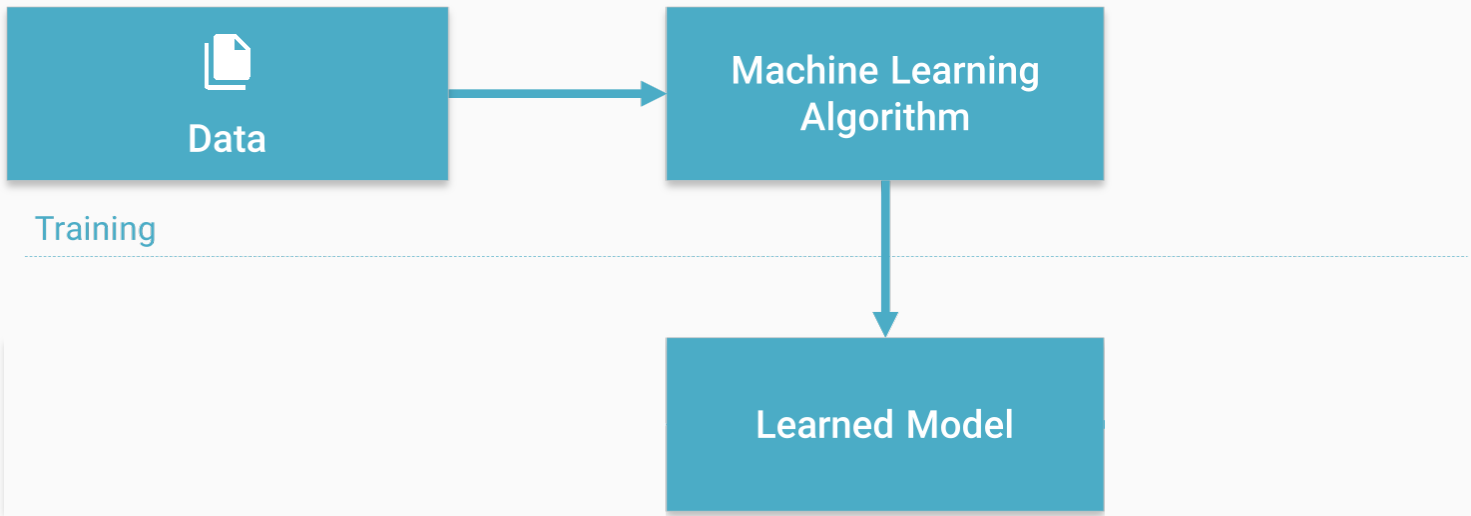
Dans notre cas, on possède des images déjà étiquetées dans différentes classes.

La méthodologie

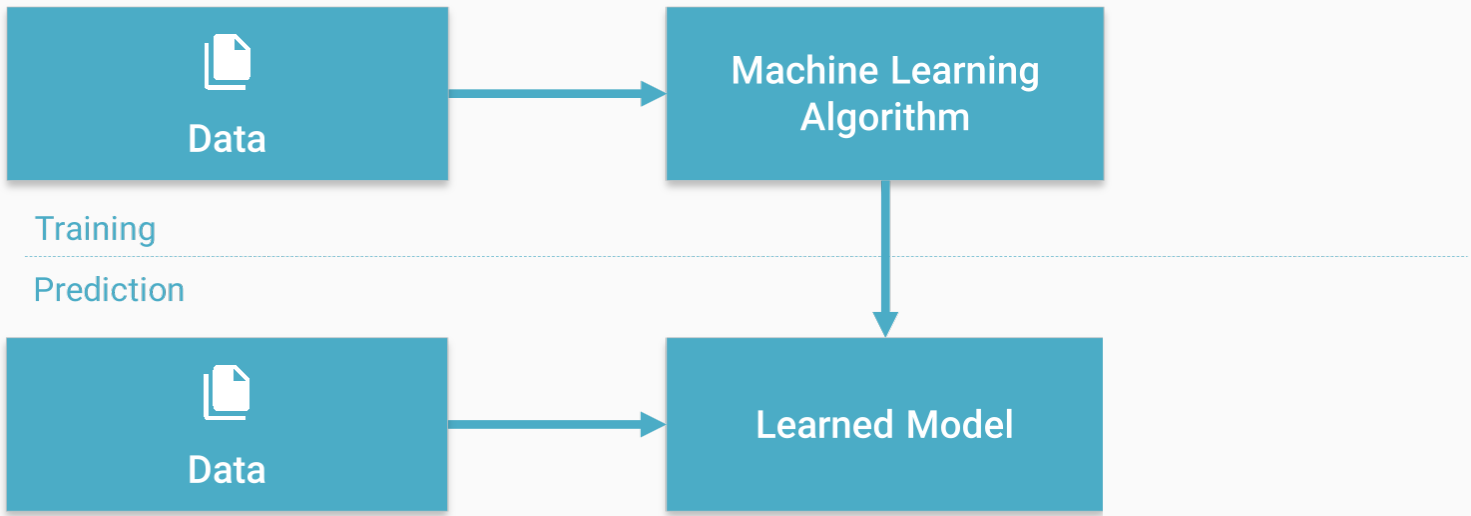


Training

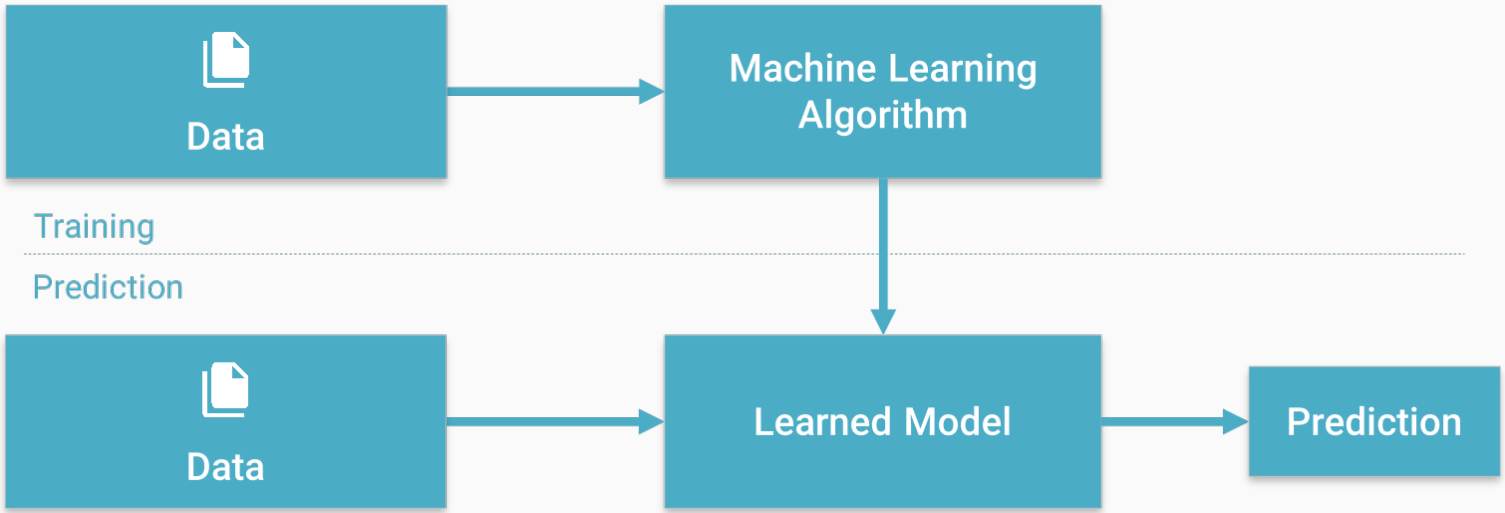
La méthodologie



La méthodologie



La méthodologie



- Pour une tâche de classification, nous avons besoin :
    - d'un ensemble de données divisé en test et entraînement

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau
  - d'un coût à minimiser

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement

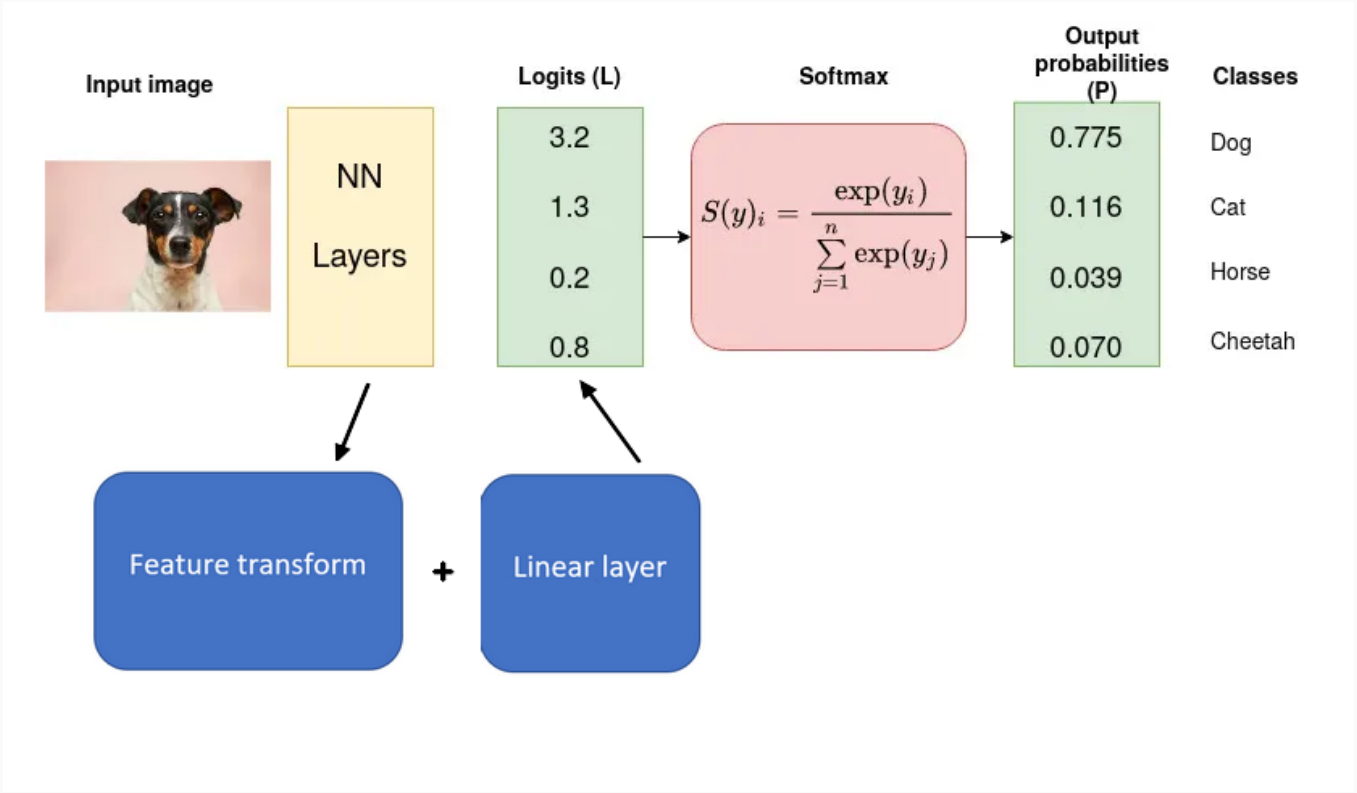
- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

Traitement de l'ensemble des données :
$\Longrightarrow$: Vu aujourd'hui en TP.
$\Longrightarrow$: Nous allons utiliser des ensembles de données déjà implémentés en Pytorch.
$\Longrightarrow$: Déjà implémenté ? Concrètement, on aura un ensemble d'images divisé en 10 classes avec chaque image attachée à une classe (via le label).
$\Longrightarrow$: Les images seront renormalisées avant d'entrer dans le réseau pour avoir des valeurs dans $[-1, 1]$ et non $[0, 1]$.

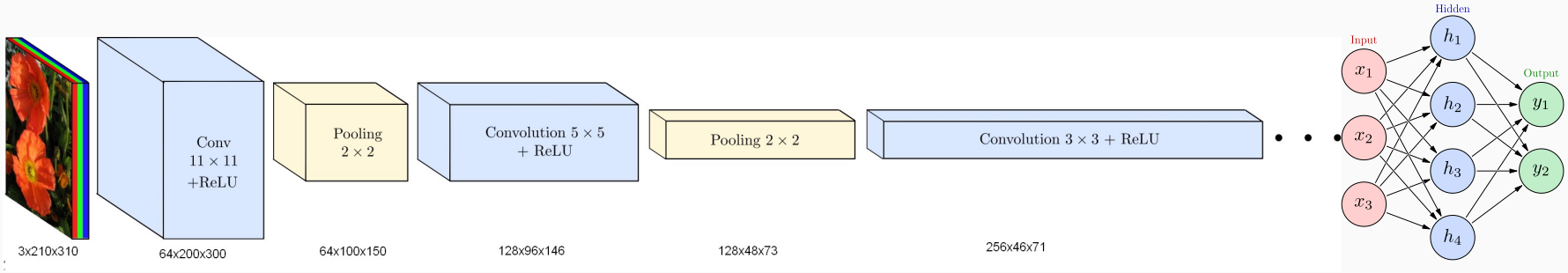
To-do list

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

Comment répondre à une tâche de classification ?



Pour les images: la feature transform via un CNN



- Une partie convolutive pour l'extraction de features.
- Une partie linéaire pour la classification des features.

La motiviation des CNN

Pourquoi un CNN ?

La motiviation des CNN

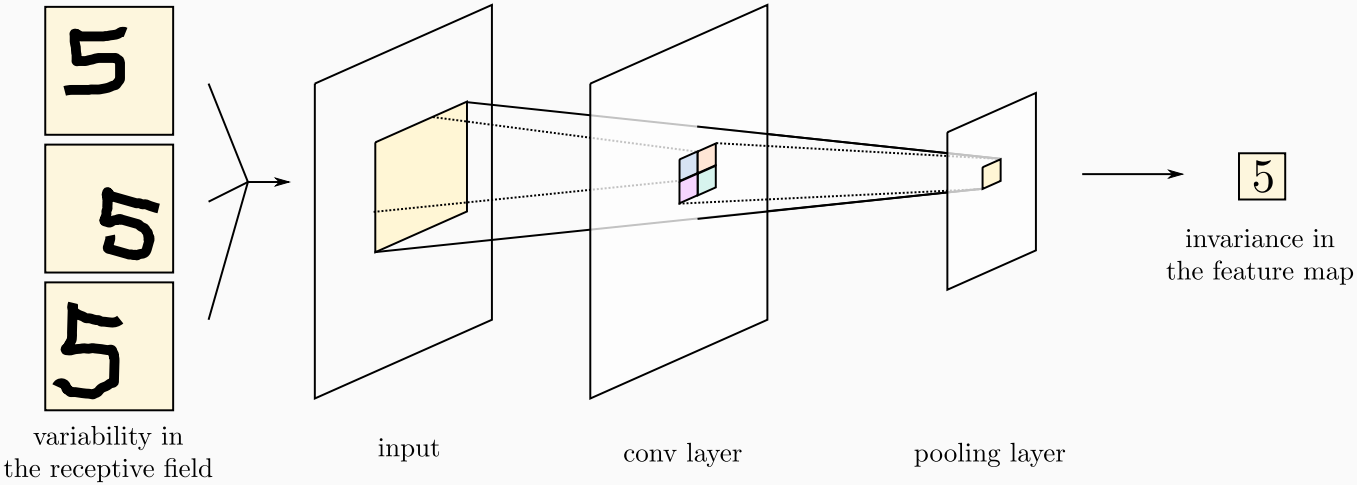
Pourquoi un CNN ?

Un CNN limite le nombre de paramètres en travaillant localement sur les images

La motiviation des CNN

Pourquoi un CNN ?

Un CNN limite le nombre de paramètres en travaillant localement sur les images



- La sortie est inchangée même avec des petits changements

Une source qui en parle : ce site (mot avec url)

Comment coder une architecture réseau ?

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 11)
        self.conv2 = nn.Conv2d(64, 128, 5)
        self.conv3 = nn.Conv2d(128, 256, 3)
        self.fc1 = nn.Linear(___, 120)  # TODO
        self.fc3 = nn.Linear(120, 10)

    def forward(self, x):
    # conv1 + Relu + pooling
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
    # conv2 + Relu + pooling
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
    # conv3 + Relu
        x = F.relu(self.conv3(x))
    # Vers les couches lineaires
        x = torch.flatten(x, 1)
    # 1ere couche lineaire + Relu
        x = F.relu(self.fc1(x))
    # 2de couche lineaire
        x = self.fc2(x)
        return x
net = Net()
```

Pourquoi fc ? Les couches linéaires sont parfois appelées "fully connected".

Comment coder une architecture réseau ?

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 11)
        self.conv2 = nn.Conv2d(64, 128, 5)
        self.conv3 = nn.Conv2d(128, 256, 3)
        self.fc1 = nn.Linear(256*46*71, 120)  # TODO
        self.fc2 = nn.Linear(120, 10)

    def forward(self, x):
        # conv1 + Relu + pooling
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        # conv2 + Relu + pooling
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        # conv3 + Relu
        x = F.relu(self.conv3(x))
        # Vers les couches lineaires
        x = torch.flatten(x, 1)
        # 1ere couche lineaire + Relu
        x = F.relu(self.fc1(x))
        #2de couche lineaire
        x = self.fc2(x)
        return x
net = Net()
```

To-do list

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?

Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?
- La maximisation de la log-vraisemblance avec l'hypothèses que les éléments de l'ensemble des données d'entraînement sont indépendantes.
- En notant $\mathcal{E}$ l'ensemble d'entraînement, le coût L s'écrit sous la forme :

$$L(\boldsymbol{W}) = \sum_{\boldsymbol{x} \in \mathcal{E}} \ell(\boldsymbol{x}, \boldsymbol{W})$$

(Voir les slides du deuxième cours)

Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?
- La maximisation de la log-vraisemblance avec l'hypothèses que les éléments de l'ensemble des données d'entraînement sont indépendantes.
- En notant $\mathcal{E}$ l'ensemble d'entraînement, le coût L s'écrit sous la forme :

$$L(\boldsymbol{W}) = \sum_{\boldsymbol{x} \in \mathcal{E}} \ell(\boldsymbol{x}, \boldsymbol{W})$$

  (Voir les slides du deuxième cours)
- Et en Python ?

Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?
- La maximisation de la log-vraisemblance avec l'hypothèses que les éléments de l'ensemble des données d'entraînement sont indépendantes.
- En notant $\mathcal{E}$ l'ensemble d'entraînement, le coût L s'écrit sous la forme :

$$L(\boldsymbol{W}) = \sum_{\boldsymbol{x} \in \mathcal{E}} \ell(\boldsymbol{x}, \boldsymbol{W})$$

  (Voir les slides du deuxième cours)
- Et en Python ?

```
criterion = nn.CrossEntropyLoss()
```

On peut choisir d'autres loss:

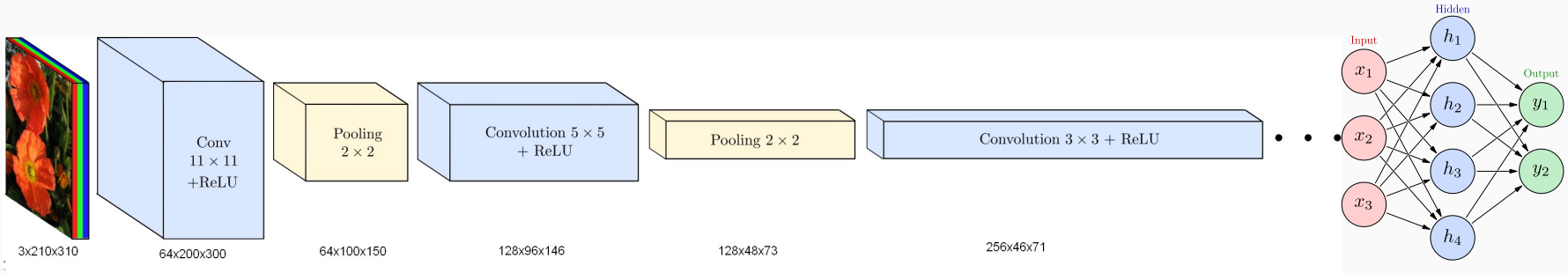
```
criterion = nn.MSELoss()
criterion = nn.L1Loss()
```

To-do list

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser ✓
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

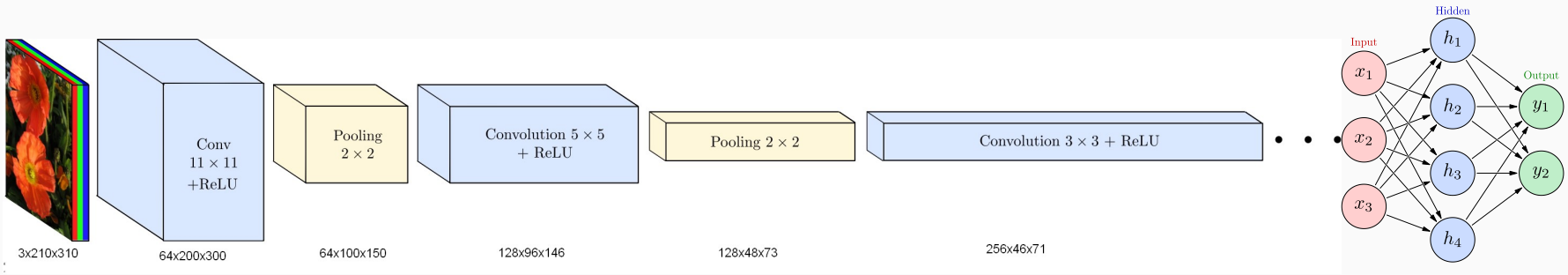
Qu'est-ce que W ?

Dans le réseau ci-dessous, que chercher-t-on à optimiser ?



Qu'est-ce que W ?

Dans le réseau ci-dessous, que chercher-t-on à optimiser ?



L'architecture du réseau ne va pas changer au cours de l'entraînement :

- La taille des noyaux
- Le nombre de canaux de sortie
- Les fonctions d'activation
- La taille des matrices de la partie linéaire

Comment optimiser le réseau ?

$$L(\boldsymbol{W}) = \sum_{\boldsymbol{x} \in \mathcal{E}} \ell(\boldsymbol{x}, \boldsymbol{W})$$

On découpe l'ensemble de données $\mathcal{E}$ en "batchs" $\mathcal{B}$ tels que $\mathcal{E} = \bigcup \mathcal{B}$

Comment optimiser le réseau ?

$$L(\boldsymbol{W}) = \sum_{\boldsymbol{x} \in \mathcal{E}} \ell(\boldsymbol{x}, \boldsymbol{W})$$

On découpe l'ensemble de données $\mathcal{E}$ en "batchs" $\mathcal{B}$ tels que $\mathcal{E} = \bigcup \mathcal{B}$

- Pour chaque époque :
  - Pour chaque batch $\mathcal{B}$ de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer $\ell(W, \boldsymbol{x})$ pour $\boldsymbol{x}$ dans le batch
    - Calculer $\boldsymbol{g} = \sum_{\boldsymbol{x} \in \mathcal{B}} \nabla_{\boldsymbol{W}} \ell(W, \boldsymbol{x})$
    - Faire un pas de SGD :

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \boldsymbol{g}$$

Comment optimiser le réseau ?

$$L(\boldsymbol{W}) = \sum_{\boldsymbol{x} \in \mathcal{E}} \ell(\boldsymbol{x}, \boldsymbol{W})$$

On découpe l'ensemble de données $\mathcal{E}$ en "batchs" $\mathcal{B}$ tels que $\mathcal{E} = \bigcup \mathcal{B}$

- Pour chaque époque :
  - Pour chaque batch $\mathcal{B}$ de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer $\ell(W, \boldsymbol{x})$ pour $\boldsymbol{x}$ dans le batch
    - Calculer $\boldsymbol{g} = \sum_{\boldsymbol{x} \in \mathcal{B}} \nabla_{\boldsymbol{W}} \ell(W, \boldsymbol{x})$
    - Faire un pas de SGD :

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \boldsymbol{g}$$

- époque = désigne un passage complet du jeu de données d'entraînement par l'algorithme.
- batch = morceau du jeu de données
- Pour cette raison, input d'un réseau de la forme $[b, c, M, N]$
- Comment est calculé le gradient ?

Pourquoi des batches ?

Avant :

- Pour chaque époque :
  - Calculer $\ell(W, \boldsymbol{x})$ pour $\boldsymbol{x}$ dans l'ensemble d'entraînement
  - Calculer $\boldsymbol{g} = \sum_{\boldsymbol{x} \in \mathcal{E}} \nabla_{\boldsymbol{W}} \ell(W, \boldsymbol{x})$
  - Faire un pas de SGD :

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \boldsymbol{g}$$

Après :

- Pour chaque époque :
  - Pour chaque batch $\mathcal{B}$ de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer $\ell(W, \boldsymbol{x})$ pour $\boldsymbol{x}$ dans le batch
    - Calculer $\boldsymbol{g} = \sum_{\boldsymbol{x} \in \mathcal{B}} \nabla_{\boldsymbol{W}} \ell(W, \boldsymbol{x})$
    - Faire un pas de SGD :

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \boldsymbol{g}$$

Pourquoi cela fonctionne ?

- La fonction de coût peut s'écrire

$$L(\boldsymbol{W}, \mathcal{E}) = \sum_{\left(\boldsymbol{x}^i, d^i\right) \in \mathcal{E}} \ell\left(\boldsymbol{x}^i, d^i\right)$$

- En supposant que les batches cartographient uniformément l'ensemble d'entraînement $\mathcal{E}$, pour un batch $\mathcal{B}$:

$$\begin{aligned}
\mathbb{E}_{\mathcal{B}}\left(\nabla_{\boldsymbol{W}_k} L(\boldsymbol{W}; \mathcal{B})\right) &= \mathbb{E}_{\mathcal{B}}\left(\sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{B}} \nabla_{\boldsymbol{W}_k} \ell(\boldsymbol{y}^i; \boldsymbol{d}^i)\right) \\
&= \mathbb{E}_{\mathcal{B}}\left(\sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{E}} \mathbf{1}_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{B}} \nabla_{\boldsymbol{W}_k} \ell(\boldsymbol{y}^i; \boldsymbol{d}^i)\right) \\
&= \frac{|\mathcal{B}|}{|\mathcal{E}|} \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}_k} \ell(\boldsymbol{y}^i; \boldsymbol{d}^i) = \frac{|\mathcal{B}|}{|\mathcal{E}|} \nabla_{\boldsymbol{W}_k} L(\boldsymbol{W}).
\end{aligned}$$

- Conclusion: Le calcul du gradient sur un batch est un estimateur non biaisé du gradient sur l'ensemble des entraînements.

Comment optimiser le réseau avec Python ?

```
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
for epoch in range(Nb_epochs):
    for i, data in enumerate(trainloader, 0): #Divise l'ensemble de
     donnees en batchs

        batch, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

To-do list

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser ✓
- Ensuite, il y a une phase d'entraînement ✓
- Enfin, il y a une phase de test

Comment vérifier la performance du réseau ?

- En utilisant l'ensemble de données de test.

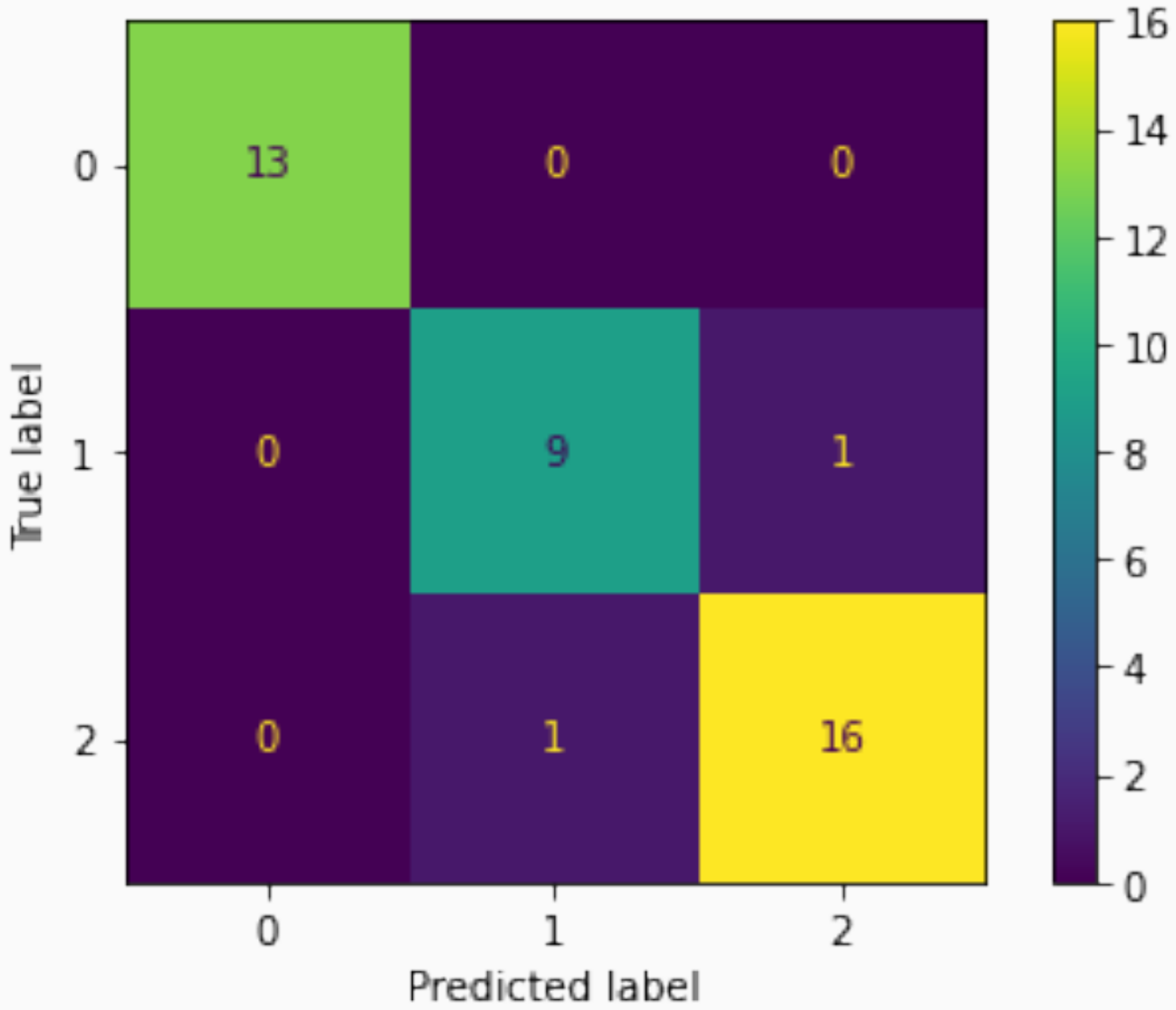
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
#Classification report
print(classification_report(t_test,pred))

#Confusion matrix
print(confusion_matrix(t_test,pred))
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(
    t_test,pred))
disp.plot()
```

Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	0.90	0.90	0.90	10
2	0.94	0.94	0.94	17
accuracy			0.95	40
macro avg	0.95	0.95	0.95	40
weighted avg	0.95	0.95	0.95	40

Les formules utilisées pour construire le tableau peuvent être trouvées à ce lien

Matrice de confusion



To-do list

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser ✓
- Ensuite, il y a une phase d'entraînement ✓
- Enfin, il y a une phase de test ✓

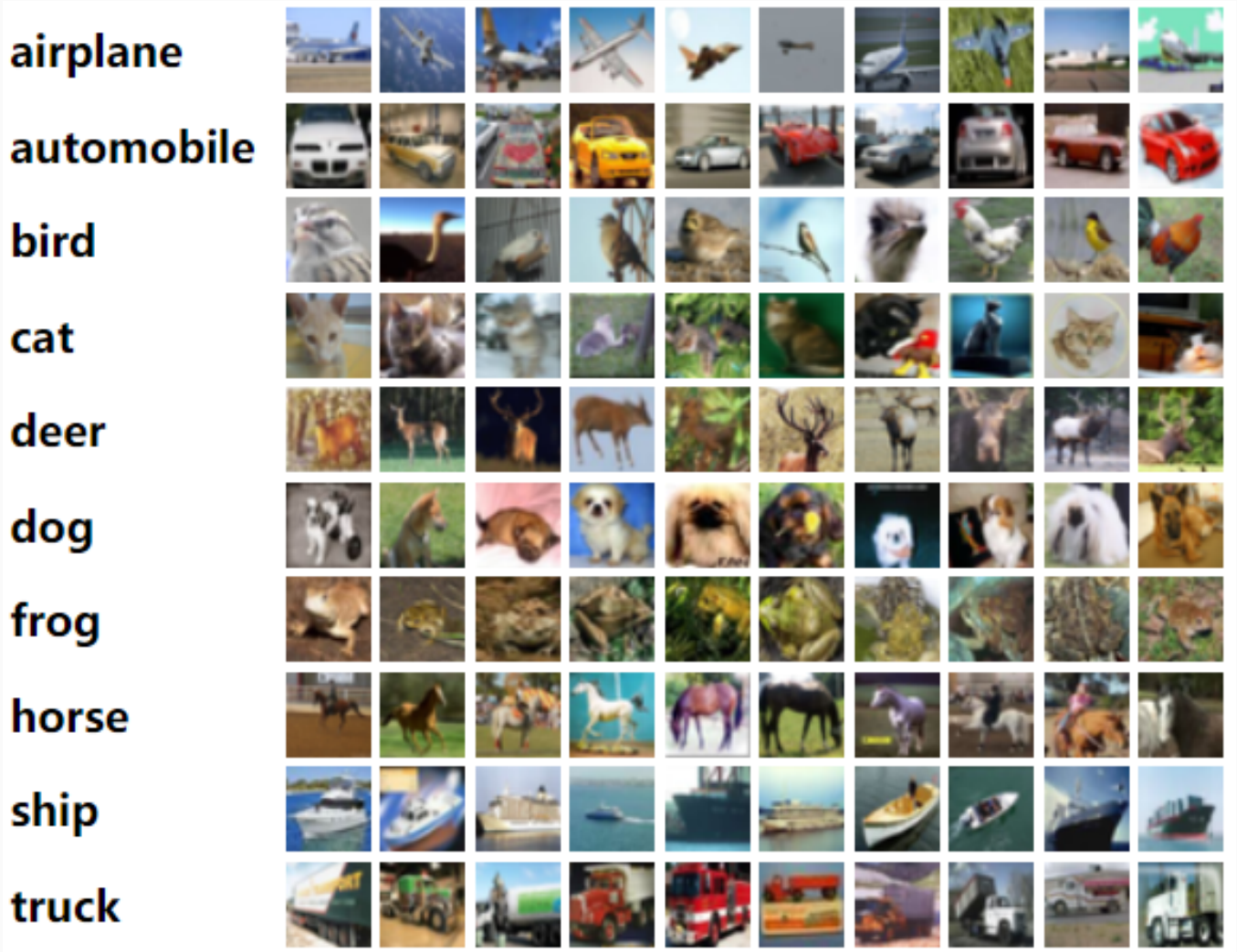
Des problèmes pour plus tard

Les problèmes que nous ne nous sommes pas posés :

- Choix de l'architecture du réseau
  $\rightarrow$ décidée avec un ensemble de validation et l'état de l'art
- Choix des hyperparamètres d'optimisation (lr,momentum)
  $\rightarrow$ décidés avec l'affichage du coût au cours du temps
- Ajout de couche "batchnorm" pour améliorer l'entraînement
  $\rightarrow$ traité en M2 avec B. Galerne

En TP aujourd'hui:

Entraînement d'un réseau pour classificier CIFAR10.



+ retour sur le TP1

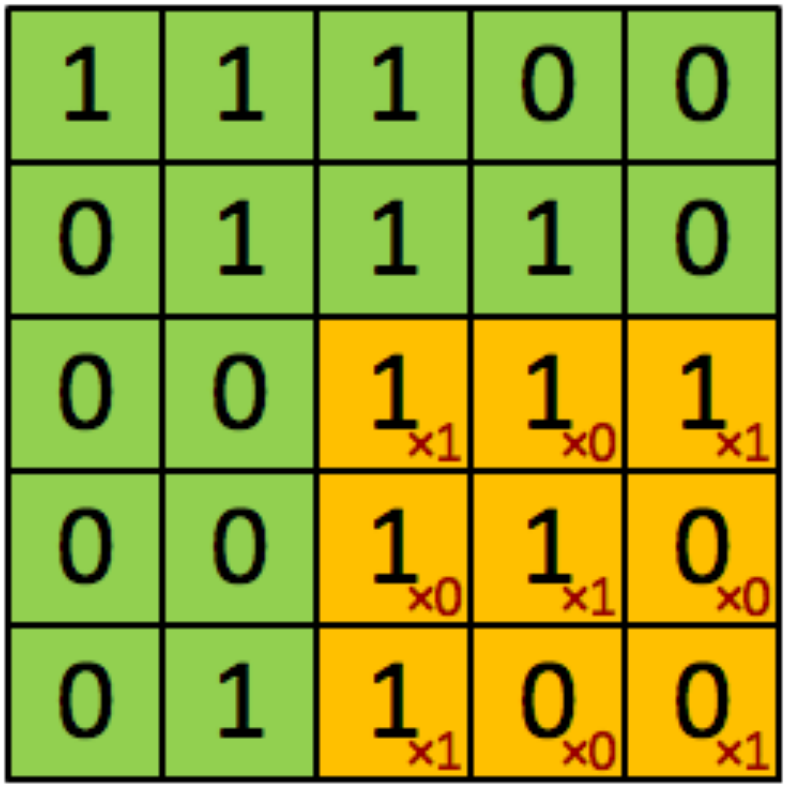
En projet:

Entraînement d'un réseau pour classifier fashion MNIST.

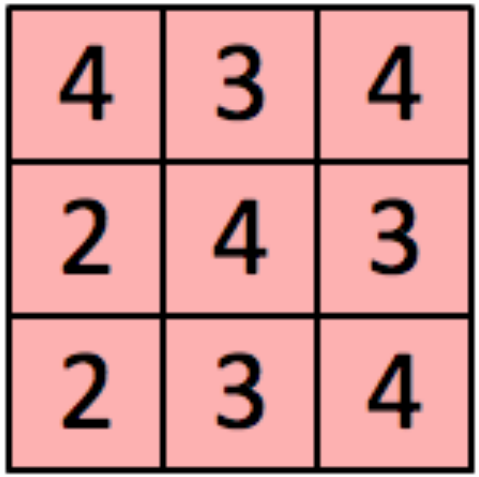
Projet : pour le **12 mai 2024**.

Questions?

Convolution

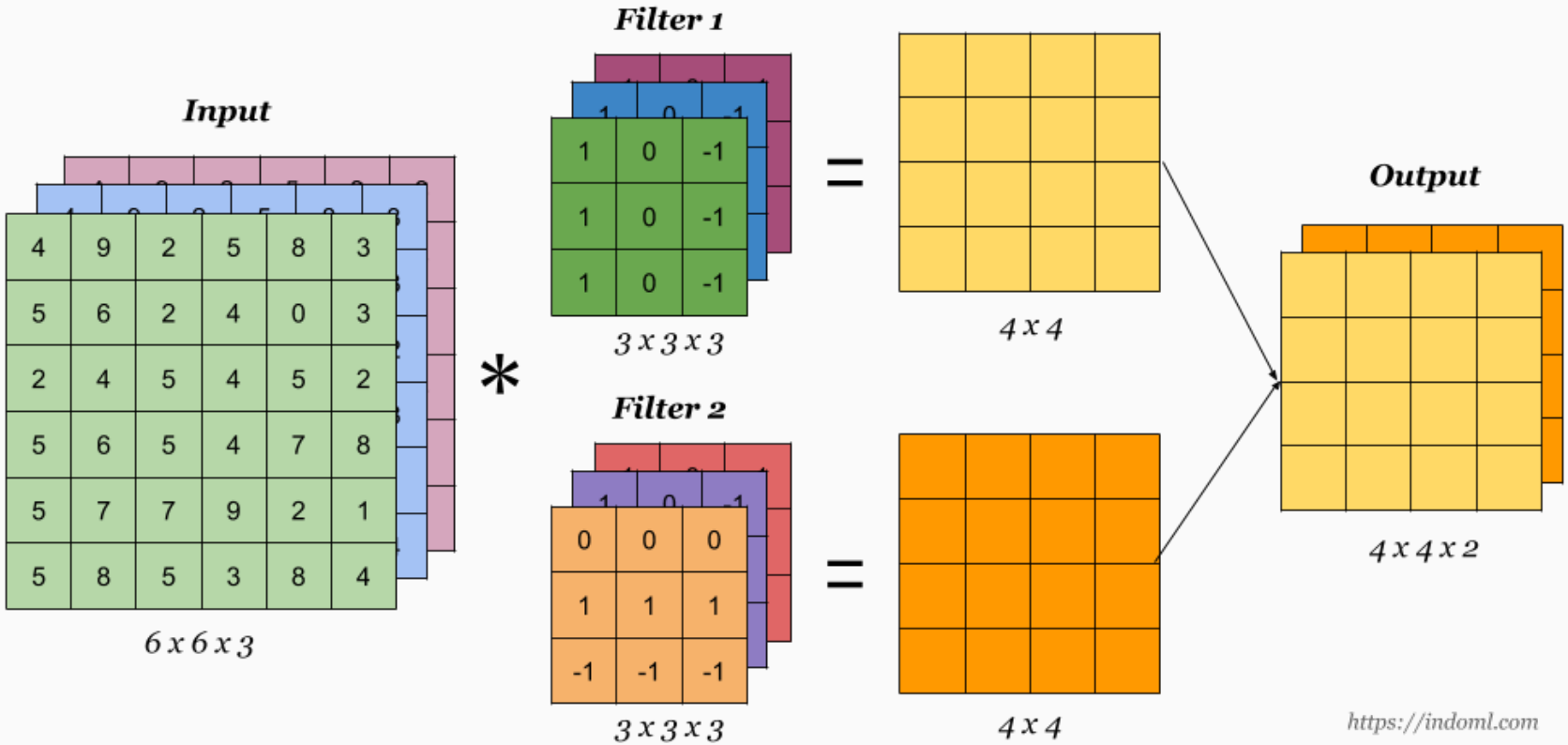


Image



Convolved Feature

Convolution avec différents canaux (sans biais)



Pooling